# Performance Evaluation of a Firm Real-Time DataBase System

Stuart Shih, Young-Kuk Kim, and Sang H. Son
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22903, USA
son@cs.virginia.edu

## Abstract

In conventional database systems, performance is primarily measured by the number of transactions completed within a unit time. In real-time applications, timing and criticality characteristics of transactions must be taken into account. In this paper, we examine the performance of StarBase, a firm real-time database system. The deadline guarantee ratio and average response times are the primary performance measures. There have been performance studies on real-time database systems, but most of them were performed using simulation. This work demonstrates the feasibility of developing a real-time database system with an acceptable performance.

## 1. Introduction

As real-time applications increase in complexity, so do their data requirements. For several years, researchers have sought a general solution to the problem of collecting, storing, and retrieving data in real-time by devising database management systems to manage data in a time-cognizant and predictable manner [Yu94]. Despite all of its features, a conventional database system is not quite capable of meeting the demands of a real-time system. Typically, its goals are to maximize transaction throughput, minimize response time, and provide some degree of fairness. A real-time database system, however, must adopt goals which are consistent with any real-time system: providing the best service to the most critical transactions and ensuring some degree of predictability in transaction processing [Son94].

In conventional database systems, all transactions should have the same opportunity to obtain system resources to help complete their execution. Since the number, not type of transactions completed in a given time unit is important, measuring throughput and average response time is an accurate way of accessing the performance of a conventional database system. In a real-time database system, timing and criticality characteristics of transactions must be taken into account, and hence a different performance measure should be used. Transactions with higher timing or criticality constraints are of greater importance to the database system and thus, should be allocated a larger share of the available computation time and resources. Given these constraints, the deadline guarantee ratio is a much more accurate measure of performance for real-time database systems. To differentiate the importance of individual transactions, real-time transactions are assigned a priority or criticality when submitted. The real-time database attempts to maximize the total value for a set of transactions by allocating system resources to transactions with the highest priority to provide the best chance of completion before their deadline expires.

In this paper, we examine the performance of a firm real-time database system, called StarBase, being developed at the University of Virginia. One of our goal is to demonstrate the feasibility of developing a real-time database system with an acceptable performance.

## 2. StarBase

StarBase is a firm real-time database system which supports the concurrent execution of transactions and seeks to minimize the number of high-priority transactions that miss their deadlines [Leh95]. StarBase uses no a priori information about the transaction workload and discards tardy transactions at their deadline points. StarBase runs on top of RT-Mach, a real-time operating system under development at Carnegie Mellon University [Tok90].

In a firm real-time setting, if a transaction cannot be completed by the database system within the given deadline time, then that transaction is discarded from the system. This is in contrast to a hard real-time database system, where a transaction missing a deadline can result in a catastrophic event, and a soft real-time database system, where there is still some value associated with completing a transaction even after its deadline has passed.

StarBase differs from previous real-time database work in that a) it relies on a real-time operating system which provides priority-based scheduling and time-based synchronization, and b) it deals explicitly with data contention and deadline handling in addition to transaction scheduling, the traditional focus of simulation studies.

0-8186-7106-8/95 $04.00 © 1995 IEEE

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. REPORT DATE<br>**1995** | 2. REPORT TYPE | 3. DATES COVERED<br>**00-00-1995 to 00-00-1995** |
| 4. TITLE AND SUBTITLE<br>**Performance Evaluation of a Firm Real-Time DataBase System** | | 5a. CONTRACT NUMBER<br>5b. GRANT NUMBER<br>5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | 5d. PROJECT NUMBER<br>5e. TASK NUMBER<br>5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**University of Virginia,Department of Computer Science,151 Engineer's Way,Charlottesville,VA,22904-4740** | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | 10. SPONSOR/MONITOR'S ACRONYM(S)<br>11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT<br>**Approved for public release; distribution unlimited** | | |
| 13. SUPPLEMENTARY NOTES | | |
| 14. ABSTRACT | | |
| 15. SUBJECT TERMS | | |

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | | **9** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

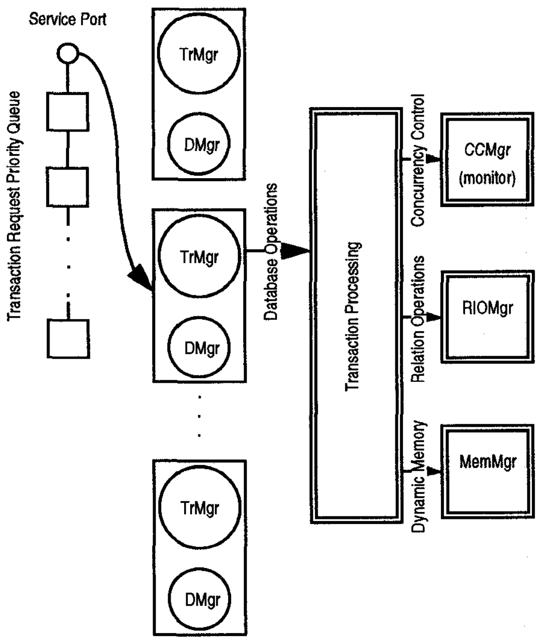

Fig. 1: StarBase Server Architecture

The design of StarBase is shown in Fig. 1.

The StarBase database system receives transaction requests from database clients and places them on a priority queue. It is assumed that database clients are physically disparate from the server, so they pass messages to communicate with the DBMS server. Transaction requests are sent via RT-Mach's Inter-Process Communication (IPC) mechanism and are queued at the server's service port. RT-Mach provides a naming service with which StarBase registers its service port during initialization. Clients look up the service port by querying the name server with StarBase's well-known name. There are a fixed number of threads (light-weight processes), called Transaction Managers, which dequeue those requests and perform the basic operations which constitute the transaction. The transaction processing unit in turn implements these basic operations. The transaction managers rely on lower-level services to obtain the resources (memory, relations, etc.) necessary for the transaction. Each resource manager must ensure that transactions access their resources in a consistent and orderly fashion.

When the transaction manager starts to service a new transaction request, a deadline thread also begins its execution. If the deadline time expires at any point before the completion of the transaction manager, then the deadline thread informs the transaction manager of the expired deadline situation. The transaction manager then terminates its execution and returns a deadline abort message to the client. Otherwise, the transaction manager informs the client with either a simple "success" response message or possibly retrieved results, depending on the transaction request submitted. During this time however, the client is blocked waiting for a message from the server. This limitation prevents the same client from submitting multiple transaction requests while a former request is still pending.

To resolve data conflicts between competing transactions, a concurrency control algorithm must be used. There are two major types of concurrency control which have been considered for use in real-time databases: lock-based and optimistic methods. In general, lock-based methods delay transactions to avoid having them access data in an inconsistent way, whereas optimistic methods abort transactions.

StarBase uses a real-time optimistic concurrency control method called WAIT-X [Har91], which has been experimentally shown to outperform lock-based concurrency control methods in a firm real-time setting. With WAIT-X, a transaction T executes unhindered until it reaches the point where it can commit (i.e., make its changes to the data permanent) and WAIT-X determines which transactions should be aborted due to the conflict with T. Unlike conventional concurrency control, WAIT-X employs a priority-cognizant commit test: If high-priority transactions comprise less than X% of all of T's conflictors, T can commit, aborting all conflictors in the process. Otherwise T waits so that higher priority transactions may proceed. It was found experimentally that low values of X tend to minimize the deadline miss ratio for light loads, and high values of X tend to minimize the deadline miss ratio for heavy loads. When X = 50% is used as the threshold value, it minimizes the overall deadline miss ratio, but applications which require minimization of the highest-priority deadline miss ratio must use a greater value for X.

StarBase uses T-Tree indexing to decrease the data conflict rate and improve response time. T-tree indexing was proposed specifically for in-memory retrieval to improve the performance over indexing mechanisms for disk-based data [Leh86]. The T-tree is a binary tree which contains multiple elements in each node. The description and implementation of the T-Tree indexing mechanism in StarBase is described in [Geo93].

## 3. Real-Time Mach

StarBase currently runs on a 486 DX2/50E machine running the RT-Mach MK83i operating system. RT-Mach, a variant of Mach, provides many of the priority-based real-time features that StarBase relies on to service transactions in a real-time environment. The RT-Mach features StarBase employs are briefly described in this section.

### 3.1. Real-Time Threads

All real-time threads in StarBase are created as aperiodic threads. Given a real-time application however, we expect that some of the transaction requests will arrive periodically, so it is a good feature that RT-Mach provides both types of threads [Tok90]. The priority field in the thread structure is used by RT-Mach for scheduling, priority handoff and priority inheritance in RT-IPC, and in RT-Sync.

### 3.2. Real-Time IPC (RT-IPC)

The manner in which the IPC mechanism in RT-Mach handles messages for StarBase can be described by three main attributes: the message queueing policy, the priority hand-off policy, and the priority inheritance policy. For StarBase, the queueing policy chosen is priority-based, so that the highest priority transaction is always serviced by the next available transaction manager. Since this message ordering does not take the deadline into account, transactions with the same priority are queued depending on their order of arrival.

The priority handoff is enabled in StarBase, allowing the transaction manager thread that reads the transaction request message to execute at the priority specified in the request message. This priority message inheritance mechanism lasts until the transaction manager thread has finished servicing the transaction request. If a higher priority thread receives a message with a lower priority, then no inheritance occurs and the thread continues to execute at the same priority. To prevent such a situation from occurring in StarBase, the transaction managers are initialized with the lowest system priority level possible. The inheritance mechanism will always then correctly raise the priority level of the transaction manager to reflect the importance of the transaction request [Kit93].

### 3.3. Real-Time Synchronization (RT-Sync)

Currently, StarBase relies on the mutual exclusion with a lock variable supplied by RT-Mach to provide synchronization between transactions during the read, validation, and write phase of the WAIT-50 scheme. To ensure that the next highest priority transaction obtains the lock object after it is released by the current transaction, the lock policy follows the basic priority inheritance protocol. This policy prevents any possible chain of blocking from middle priority transactions and guarantees that the lower priority transaction is serviced quickly so that the lock can be released [Tok91].

### 3.4. Real-Time Scheduling

RT-Mach supports many different scheduling policies, but all RT-Mach scheduling policies other than the Mach timesharing policy causes the MK83i version of the RT-Mach operating system to halt after executing StarBase for a short time. As a result, the priorities of the transaction manager threads in StarBase are not taken into account by the Mach scheduler, and threads receive equal time quantums during execution of transaction requests. Ideally, StarBase should incorporate the fixed-priority round robin scheduling policy, which allocates a higher time quantum of execution for threads with higher priorities [Tok90].

## 4. StarBase Transaction Workload Generator

The workload generator is modelled after previous work on two-phase locking (2PL) and optimistic concurrency control (OCC) [Lee95, Hua91]. In each of these studies, the workload generator provides adjustable runtime parameters to alter the workload submitted to the database system. This section describes how the StarBase transaction workload generator creates different workload and environments to test the performance of StarBase.

### 4.1. Test Environment

The workload created by the StarBase workload generator can simulate incoming transactions for two different types of real-time application areas. The first area is information management systems, where transactions arrive aperiodically and consecutive transactions have a certain arrival period between each other. In such a system, the number of users is fixed, and each user submits a new transaction only after the previous one has completed. These transactions may or may not have timing constraints, and therefore have soft deadlines. An airline reservation system is a good example of this application area [Hua91].

The other application area is the real-time process control systems. Unlike transactions in information management systems, these transactions have hard timing constraints that must be met to prevent a catastrophic event from occurring. Transactions in such a system arrive periodically and with hard real-time constraints. Network management, traffic control, and nuclear power plants all fall in this application area [Kim95].

Since StarBase is a firm real-time database, it is reasonable to ask why a hard real-time and soft real-time environment is being used to test performance. This is the first time StarBase has undergone any rigorous performance testing. To accurately determine how StarBase performs, it was decided to examine StarBase under both real-time environments. The results from performance testing would be applied to the future development of StarBase in possibly one of these two application areas, or for the area in which different types of transactions (soft, firm, and hard) are all mixed.

## 4.2. Workload Transactions

Transactions in a workload consist of either read (select) or read/write (update) operations. The number of select and update operations in a workload is modified using the probability of write parameter Pw. As this probability increases, the possibilities of data conflicts between transactions increases as well. The length of a transaction is determined using the tuples accessed parameter. The tuples accessed by a transaction can be an explicit number or an interval range, where the number of tuples is randomly chosen from the given range of numbers.

Each of the transactions in a workload performs an operation on relational tables.These tables must be defined in advance so that their schema information can be used in the creation of transactions in the workload. The sole requirement for tables used in the transaction generation for a workload is a primary key attribute field of float type. The primary key serves as both a possible attribute field for indexing and also as a way for transactions to access different tuples within a relational table.

## 4.3. Transaction Deadline Calculation

As the transaction generator creates new transactions, a deadline function calculates an estimated deadline execution time for the transaction. This deadline time for the transaction is dependent on the type of database operation, the tuples accessed, and memory copies of results to the message buffer or to temporary storage. At the present time, the deadline function accurately calculates deadline times for select and update operations that access relational tables given the following constraints:

- All attributes besides the primary key are character data types and of equal length
- A relation table has no more than 1000 tuples
- The total bytes of all the attribute fields in the relation is no more than 800 bytes
- All select and update operations have a where clause of the form

"where primary_key >= a & primary_key < b"
where a and b are constants and $a < b$

The calculated deadline time is not total execution time, but rather the average expected time to complete a transaction assuming no other transactions are executing in the system. This time also does not include the required message passing time for sending the transaction message and receiving any results.

The timing results for select operations on 200 to 1000 tuple relation tables with different total attribute bytes show that the select deadline time, SDT, can be calculated using the following equation:

$$SDT = SIT + ((TTA / 200) * Tr)$$

SIT = Select Initialization Time
Tr = Total time to read 200 tuples from a table
TTA = Total tuples accessed

For tables with total tuples ranging from 0 to 1000 tuples, there is a direct relation between time and tuples read. Tables with 400 tuples have a Tr twice as large as the 200 tuple base case, tables with 800 tuples have a Tr four times as large, and so on. Tr increases as the total attributes bytes for a tuple goes up. This increase is also linear and can be calculated from extrapolation of the Tr values for the 200 and 1000 tuple relation tables.

The update deadline time, UDT, is similar to the SDT equation with one exception. Whereas the number of attributes in a relation does not affect the select operation, the update operation must take this into account. A table that has 400 tuples with 1 attribute field will have different update times than a table that has 200 tuples with 2 attribute fields. Each extra attribute adds a constant time that is included in the modification of the SDT equation show below:

$$UDT = UIT + ((TTA / 200) * Tu)$$
$$Tu = Tbu + ((TA - 1) * Tx)$$

UIT = Update Initialization Time
Tu = Total time to update 200 tuples from a table
TTA = Total tuples accessed
Tbu = Base update time for 200 tuples from a table
TA = Total attributes updated
Tx = Extra Time for additional attribute

Timing results for update operations on 200 to 1000 tuple relation tables with different total attribute bytes confirm the formula above. In Figure 2, the increase in update times due to the increase in the number of attributes is shown.

## 4.4. Transaction Priority Calculation

Longer length transactions are more likely to miss their deadlines due to the increased probability of data conflicts as they access more tuples in the relation. To avoid the starvation problem mentioned in [Hua91], priorities are assigned according to the number of tuples each transaction accesses. Given that thread priorities are considered in IPC, synchronization, and scheduling in RT-Mach, longer transactions with higher priorities have a better chance to complete. The following equation is used to calculate the priority for a given transaction:

$p = min\_priority - (TTA / TT) * (PR)$

min_priority = 31
max_priority = 0
PR = priority range = (min_priority - max_priority)
TTA = total tuples accessed
TT = total tuples in the relation table

## 4.5. Transactions in Information Systems

Eight StarBase clients are initially created when the StarBase workload generator is executed. The number of clients that are actually used is a parameter that can be changed. Once a transaction workload is executed in this environment, the client attempts to service the next transaction request stored in the workload. A client, upon successfully obtaining one of the transaction requests from the workload, sends the request to the StarBase server, waits for the server to reply, and repeats this process until all transactions in the workload have been serviced.

## 4.6. Transactions in Process Control Systems

To simulate periodic transactions, the StarBase workload generator employs periodic threads. Each thread has a client session through which it can immediately submit a transaction request. After each thread period, the next instantiation of the thread submits the next transaction request in the workload regardless of whether the current thread instantiation has completed. The StarBase workload generator currently permits a maximum of 8 periodic threads to be created in this test environment.

## 4.7. Adjustable Run-Time Parameters

The parameters used strictly for the simulation of information systems are the multiprogramming level and external think time. With the multiprogramming level parameter, the total number of clients that can send a transaction request to the StarBase server at any time is limited to the MPL level. This parameter allows us to simulate an environment with a fixed number of users. The external think time, on the other hand, models the elapsed time between submission of consecutive transaction requests by the same user.

The period window factor parameter is only relevant to the simulation of open process systems. In the open process system environment, the same transactions arrive periodically, so we are more interested in how the database performs given a particular transaction arrival rate. The period window factor for the workload generator controls this rate. After every period window interval, a new transaction in the workload is sent to the StarBase server. Table 1 lists the parameters used to create specific transaction workloads.

**Table 1: Workload Transaction Parameters**

| Parameter | Settings |
|---|---|
| Multiprogramming Level (MPL) | 1 - 12 |
| Write Probability (Pw) | 0.0 - 1.0 |
| Tuples Accessed | 1- Max |
| Deadline Window Factor (a) | > 0.0 |
| Period Window Interval | > 0.0 |
| Priority | 0 - 31 |
| External Think Time | >= 0 |

## 5. Experimental Results

### 5.1. Experiment 1: Resource/Data Contention

In this experiment, we examine the performance of StarBase under resource and data contention. The deadline guarantee ratio is shown in Figures 3 and 4 for updating/selecting 5 tuples from 200, 600, and 1000 tuple size relation tables. In Figures 3 and 4, the deadline guarantee ratio drops as Pw increases and the size of the table increases. This is a result of the data contention that occurs with the increase of Pw or the table size.

Although only 5 tuples are being selected/updated, in a non-indexed table it is necessary to examine all the tuples in the table during a database operation. All the tuples in the read set are then marked, and the opportunity for data conflict between read/write transactions is greatly increased for larger relation tables as write transactions take a longer amount of time searching for appropriate tuples to update.

## 5.2. Experiment 2: Transaction Priority

To test how StarBase handles transaction priorities, we examine how StarBase performs when given mixed transaction workloads. In each of these workloads, we vary the write probability Pw again, but we allow the tuples updated/selected to be 5, 25, or 45 tuples. Figure 5 illustrates the deadline guarantee ratio for mixed transaction workloads. The results show that longer transactions (i.e. those transactions that update/select more tuples) have a lower deadline guarantee ratio than shorter length transactions. As the execution time increases, the possibility of data conflict increases as well.

Given enough short transactions, we may eventually get a "transaction starvation" [Hua91] situation where long transactions are constantly being restarted because of conflicts with short transactions. The starvation problem can be avoided by assigning long transactions a higher priority. The results of running the same workload with the new priority assignment is shown in Figure 6. Under this new priority assignment, we found that the priority level, not the length of transactions, determines the deadline guarantee ratio.

## 5.3. Experiment 3: Indexing

To examine the performance of the StarBase T-Tree indexing mechanism, we execute the same workloads from Experiment 1, but using an indexed primary key and indexed transaction deadlines for the three different relation table size. We expect to see the deadline guarantee ratio stay level as Pw increases since only the specific tuples selected/updated are marked in the transaction's read and write set. The drop in the deadline guarantee ratio shown in Figures 7 and 8 can be attributed not to data conflicts, but to blocking. In the StarBase T-Tree implementation, the transaction accessing tuples from the T-Tree must lock the root of the tree. No other transactions can access the T-Tree at this time, regardless of the operation being performed on the tree. As Pw increases, update operations which lock the T-Tree for a longer period will decrease the overall deadline guarantee ratio for the entire transaction set.

We have attempted to enhance the performance of the T-Tree indexing mechanism by altering the current locking strategy. As long as the T-Tree is not altered, then transactions are able to concurrently access the T-Tree. For a write operation, which alters the tree, the lock is once again placed on the root of the tree. The performance of the new locking strategy is compared to the former one in Figure 9. For a low resource contention level, a = 4, the new locking strategy increases the number of transactions meeting their deadlines. However, further tests show that as the resource contention level increases, the blocking caused by the write operations overrides any performance advantage between the two locking strategies.

The long blocking times from write operations also conflict with transaction priority. In Experiment 2, when a mixed priority transaction workload was submitted to the StarBase server, transactions with higher priority showed better deadline guarantee ratios. With all the transactions now competing for access to the T-Tree, long update transactions cause many more shorter length transactions to miss their deadlines. Figure 10 shows the performance results of the system running a mixed transaction workload where transactions have the same priority level. As a result of the long blocking time on the T-Tree for update operations, longer length transactions actually have higher deadline guarantee ratios than short transactions. The same result occurs if shorter length transactions are given higher priorities than the longer length transactions.

## 6. Conclusions

This paper presents the design and implementation of a firm real-time database system called StarBase, running on a real-time operating system kernel, RT-Mach. We discuss how performance was evaluated in StarBase using the StarBase workload generator. By adjusting many of the run-time parameters provided by the workload generator, we have examined the performance of StarBase under different environments and different transaction workloads. In the work reported in this paper, we have taken an in-depth look at how StarBase handles resource and data contention, the consideration of transaction priorities, and the efficiency of the T-Tree indexing mechanism.

From the resource and data contention experiments, we can see that in an environment with high data contention, an indexing mechanism must be used. Searching though every tuple in a relation is clearly unacceptable in a range query, leading to both increased response time and increased data conflict rates. An improved T-Tree indexing scheme for StarBase should help here.

The results of the transaction priority tests are encouraging. Even without the proper real-time scheduling policy in place, transactions with higher priority display a higher deadline guarantee ratio. This increase occurs despite the fact that the higher priority transactions in the workloads are longer length transactions and have the same CPU time quantum as shorter-length transactions. Thus, to avoid the starvation problem mentioned in Section 4, longer-length transactions can simply be assigned higher priorities.

Even though the T-Tree indexing mechanism did not perform as well as expected, we believe that the performance can still be improved by a different locking scheme. The locking scheme should take the priority of transactions into account, and unlike the previous locking schemes, it should allow as many transactions to access the T-Tree as possible. Such a solution will not only reduce the number of transactions missing their deadlines due to T-Tree blocking, but also remove the transaction priority problem mentioned in Section 5. We plan to implement more advanced locking schemes in StarBase, and measure the performance improvements.

Future work should address how StarBase performs in an open process systems environment. Currently, RT-Mach creates periodic threads only after the instantiation of the previous thread has completed. This problem has prevented any experiments on how StarBase performs under different transaction arrival rates. We also plan to correct the fixed-policy round-robin scheduling problem so that we can obtain a truly accurate representation of how StarBase performs under a priority-driven scheduling policy.

## Acknowledgment

This work is supported in part by ONR.

## References

[Geo93] Geroge, D., "Implementation of Indexing and Concurrency Control Mechanism in a Real-Time Database," *M.S. Project*, University of Virginia, May 1993.

[Har91] Haritsa, J., "Transaction Scheduling in Firm Real-Time Database Systems," *Ph.D. thesis*, University of Wisconsin-Madison, August 1991.

[Hua91] Huang, J., "Real-Time Transaction Processing: Design, Implementation, and Performance Evaluation," *Ph.D. thesis*, University of Massachusetts at Amherst, May 1991.

[Kim95] Kim, Y., "Predictability and Consistency in Real-Time Transaction Processing," *Ph.D. thesis*, University of Virginia, May 1995.

[Kit93] Kitayama, T., T. Nakajima, and H. Tokuda, "RT-IPC: An IPC Extension for Real-Time Mach." *Proceedings of the Second Microkernel Workshop*, Sep 1993.

[Lee95] Lee, J and S. H. Son, "Performance of Concurrency Control Algorithms in Real-Time Database Systems," *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, V. Kumar (ed), Prentice Hall, 1995.

[Leh87] Lehman, T. and M. Carey, "A Recovery Algorithm for a High Performance Memory-Resident Database System," *ACM SIGMOD Conference*, May 1987.

[Leh95] Lehr, M., Y. Kim, and S. H. Son, "Managing Contention and Timing Constraints in a Real-Time Database System," *16th IEEE Real-Time System Symposium*, Pisa, Italy, Dec. 1995.

[Son94] Kim, Y., M. Lehr, D. George, and S. H. Son, "A Database Server for Distributed Real-Time Systems: Issues and Experiences," *IEEE Workshop on Parallel and Distributed Real-Time Systems*, Cancun, Mexico, April 1994, 66-75.

[Tok90] Tokuda, H., T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System," *Proceedings of the First USENIX Mach Workshop*, Oct 1990.

[Tok 91] Tokuda, H. and T. Nakajima, "Evaluation of Real-Time Synchronization in Real-Time Mach," *Proceedings of the Second USENIX Mach Workshop*, Oct 1991.

[Yu94] Yu, P., K. Wu, K. Lin, and S. H. Son,. "On Real-Time Databases: Concurrency Control and Scheduling," *Proceedings of IEEE, Special Issue on Real-Time Systems*, vol. 82, no. 1, Jan. 1994.

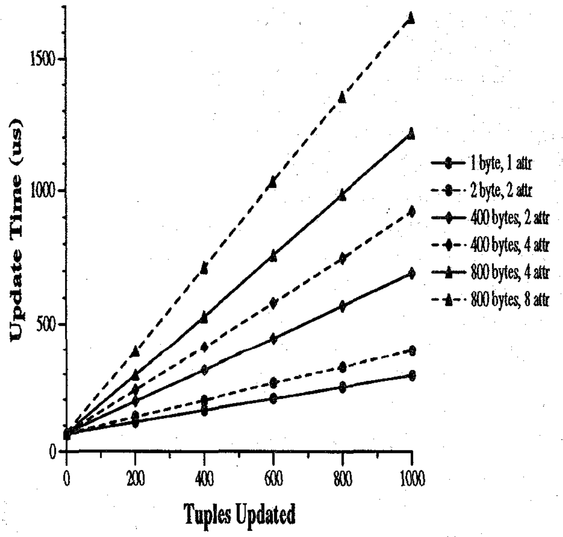

Fig. 2: Update time with different attribute number

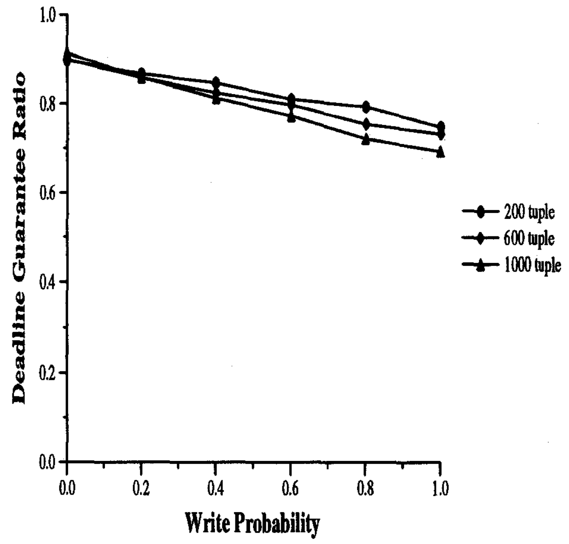

Fig. 3: Resource/data contention, MPL = 4, a = 8.0, tuples = 5

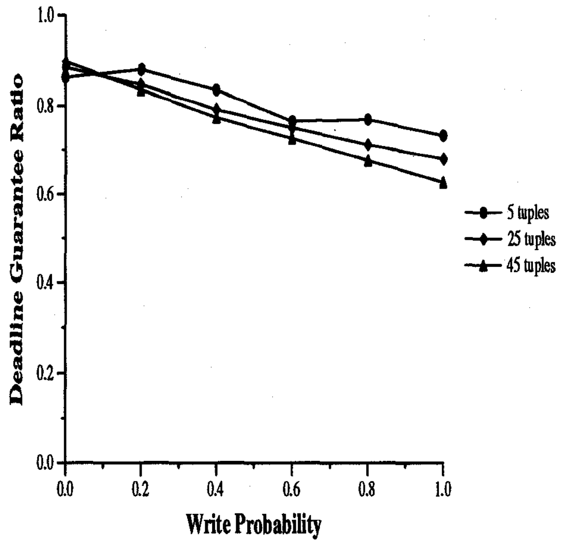

Fig. 5: Mixed transactions, MPL = 4, a= 8.0, same priority

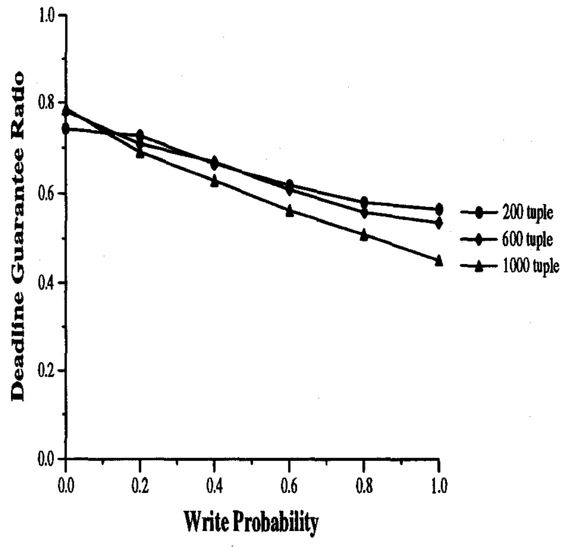

Fig. 4: Resource/data contention, MPL = 8, a = 8.0, tuples = 5

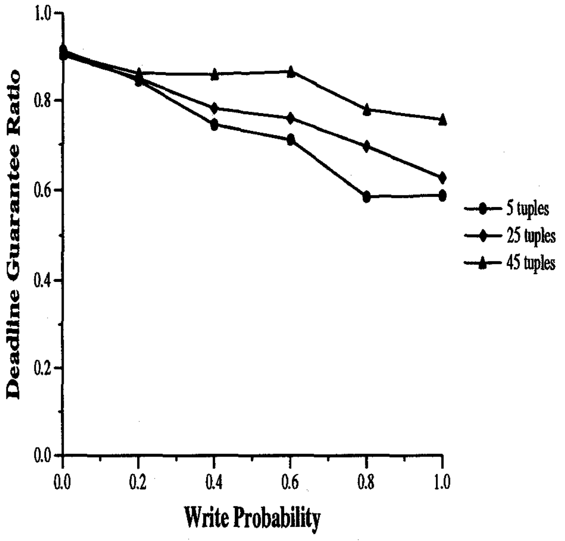

Fig.6: Mixed transactions, MPL = 4, a = 8.0, higher priority for longer transactions

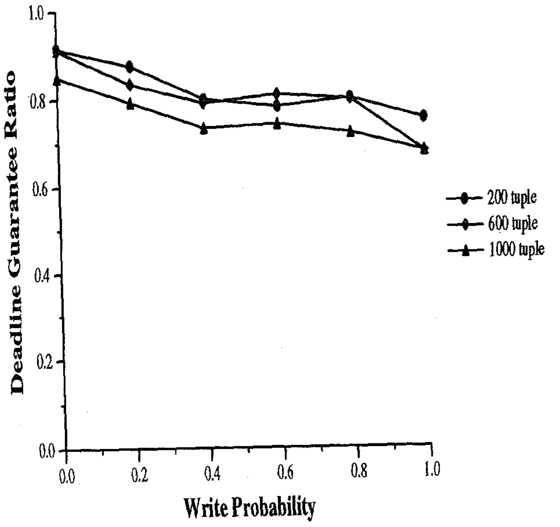

Fig. 7: Resource/data contention, MPL = 4, a = 8.0, tuples = 5, indexed

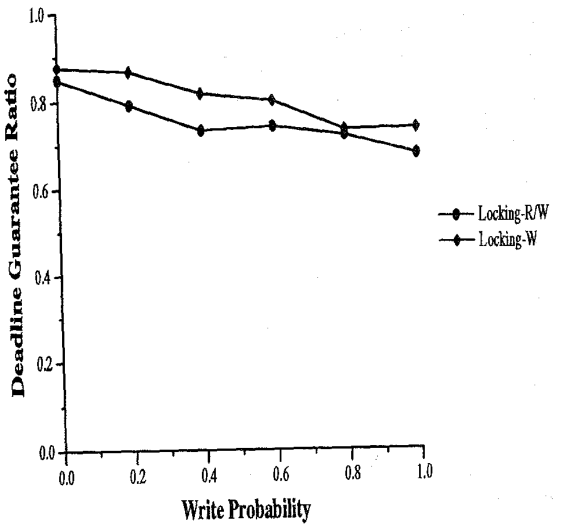

Fig. 9: Comparison of locking strategies, MPL = 4, a = 8.0, tuples = 5, table size = 600

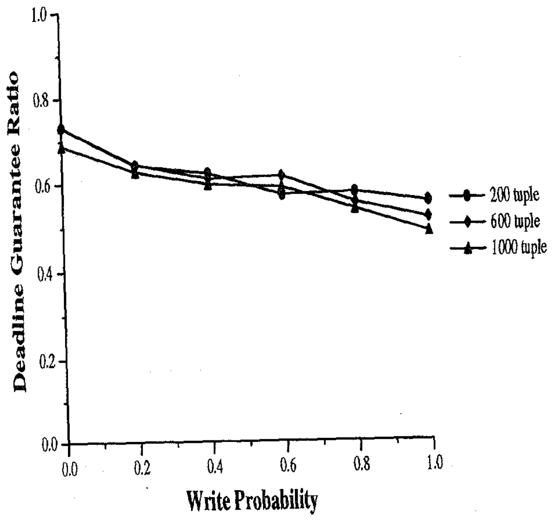

Fig. 8: Resource/data contention, MPL = 8, a = 8.0, tuples = 5, indexed

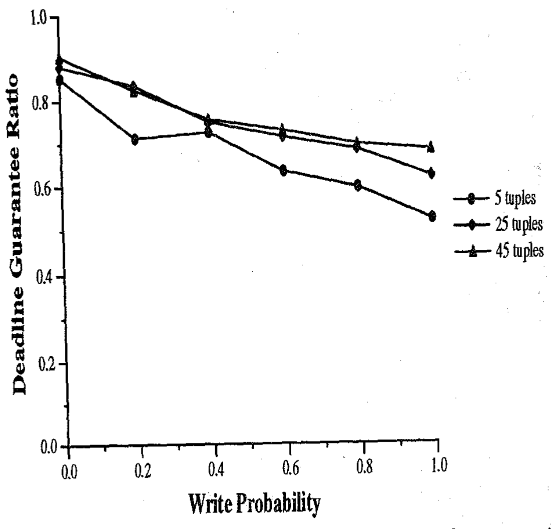

Fig. 10: Mixed transactions, MPL = 4, a = 8.0, same priority, indexed